



# GALAHAD

# FDC

USER DOCUMENTATION

GALAHAD Optimization Library version 5.1

## 1 SUMMARY

Given an under-determined set of linear equations/constraints  $\mathbf{a}_i^T \mathbf{x} = b_i$ ,  $i = 1, \dots, m$  involving  $n \geq m$  unknowns  $\mathbf{x}$ , this package **determines whether the constraints are consistent, and if so how many of the constraints are dependent**; a list of dependent constraints, that is, those which may be removed without changing the solution set, will be found and the remaining  $\mathbf{a}_i$  will be linearly independent. Full advantage is taken of any zero coefficients in the vectors  $\mathbf{a}_i$ .

**ATTRIBUTES — Versions:** GALAHAD\_FDC\_single, GALAHAD\_FDC\_double. **Uses:** GALAHAD\_CLOCK, GALAHAD\_SYMBOLS, GALAHAD\_STRING, GALAHAD\_SMT, GALAHAD\_ROOTS, GALAHAD\_SLS, GALAHAD\_ULS, GALAHAD\_SPECFILE, GALAHAD\_SPACE. **Date:** August 2006. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003. **Parallelism:** Some options may use OpenMP and its runtime library.

## 2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the USE statement

```
USE GALAHAD_FDC_single
```

with the obvious substitution GALAHAD\_FDC\_double, GALAHAD\_FDC\_quadruple, GALAHAD\_FDC\_single\_64, GALAHAD\_FDC\_double\_64 and GALAHAD\_FDC\_quadruple\_64 for the other variants.

If it is required to use more than one of the modules at the same time, the derived types SMT\_type, FDC\_time\_type, FDC\_control\_type, and FDC\_inform\_type (Section 2.3) and the subroutines FDC\_initialize, FDC\_find\_dependent, FDC\_terminate (Section 2.4) and FDC\_read\_specfile (Section 2.6) must be renamed on one of the USE statements.

### 2.1 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords REAL(rp\_) and INTEGER(ip\_), where rp\_ and ip\_ are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default REAL for the single precision versions, DOUBLE PRECISION for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to rp\_ = real32, rp\_ = real64 and rp\_ = real128 respectively as defined by the fortran iso\_fortran\_env module. The latter are default (32-bit) and long (64-bit) integers, and correspond to ip\_ = int32 and ip\_ = int64, respectively, again from the iso\_fortran\_env module.

### 2.2 Parallel usage

OpenMP may be used by the GALAHAD\_FDC package to provide parallelism for some solvers in shared memory environments. See the documentation for the GALAHAD package SLS for more details. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of -openmp). The number of threads may be controlled at runtime by setting the environment variable OMP\_NUM\_THREADS.

MPI may also be used by the package to provide parallelism for some solvers in a distributed memory environment. To use this form of parallelism, MPI must be enabled at runtime by using the correct compiler flag (usually some variant of -lmpi). Although the MPI process will be started automatically when required, it should be stopped by the calling program once no further use of this form of parallelism is needed. Typically, this will be via statements of the form

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

```
CALL MPI_INITIALIZED( flag, ierr )
IF ( flag ) CALL MPI_FINALIZE( ierr )
```

The code may be compiled and run in serial mode.

## 2.3 The derived data types

Three derived data types are accessible from the package.

### 2.3.1 The derived data type for holding control parameters

The derived data type `FDC_control_type` is used to hold controlling data. Default values may be obtained by calling `FDC_initialize` (see Section 2.4.1), while components may also be changed by calling `GALAHAD_FDC_read_spec` (see Section 2.6.1). The components of `FDC_control_type` are:

`error` is a scalar variable of type `INTEGER(ip_)`, that holds the stream number for error messages. Printing of error messages in `FDC_find_dependent` and `FDC_terminate` is suppressed if `error`  $\leq 0$ . The default is `error` = 6.

`out` is a scalar variable of type `INTEGER(ip_)`, that holds the stream number for informational messages. Printing of informational messages in `FDC_find_dependent` is suppressed if `out`  $< 0$ . The default is `out` = 6.

`print_level` is a scalar variable of type `INTEGER(ip_)`, that is used to control the amount of informational output which is required. No informational output will occur if `print_level`  $\leq 0$ . If `print_level` = 1, basic statistics of the performance of the package will be produced. If `print_level`  $\geq 2$ , this output will be increased to provide details such as the size of each neglected pivot. The default is `print_level` = 0.

`max_infeas` is a scalar variable of type `REAL(rp_)`, that holds the largest permitted infeasibility for a dependent constraint. Specifically, if  $\mathbf{x}$  satisfies  $\mathbf{a}_i^T \mathbf{x} = c_i$  for the constraints deemed to be linearly independent, it is required that  $|\mathbf{a}_i^T \mathbf{x} - c_i| \leq \text{max\_infeas}$  for those classified as dependent. The default is `max_infeas` =  $u^{1/3}$ , where  $u$  is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_FDC_double`).

`pivot_tol` is a scalar variable of type default `REAL(rp_)`, that holds the relative pivot tolerance used by the matrix factorization when attempting to detect linearly dependent constraints. See the documentation for the packages `SLS` and `ULS` for details. The default is `pivot_tol` = 0.5.

`use_sls` is a scalar variable of type default `LOGICAL`, that should be set `.TRUE.` if the `GALAHAD` package `SLS` is to be used to detect linearly dependent constraints, or `.FALSE.` if the `GALAHAD` package `ULS` is to be used instead. The default is `use_sls` = `.FALSE..`

`scale` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if the rows of  $\mathbf{A}$  are to be scaled to have unit (infinity) norm and `.FALSE.` otherwise. The default is `scale` = `.FALSE..`

`space_critical` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if space is critical when allocating arrays and `.FALSE.` otherwise. The package may run faster if `space_critical` is `.FALSE.` but at the possible expense of a larger storage requirement. The default is `space_critical` = `.FALSE..`

`deallocate_error_fatal` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if the user wishes to terminate execution if a deallocation fails, and `.FALSE.` if an attempt to continue will be made. The default is `deallocate_error_fatal` = `.FALSE..`

`symmetric_linear_solver` is a scalar variable of type default `CHARACTER` and length 30, that specifies the external package to be used to solve any symmetric linear system that might arise. Current possible choices are 'sils', 'ma27', 'ma57', 'ma77', 'ma86', 'ma97', 'ssids', 'pardiso' and 'wsmp', although only 'sils' and, for OMP 4.0-compliant compilers, 'ssids' are installed by default. See the documentation for the `GALAHAD` package `SLS` for further details. The default is `symmetric_linear_solver` = 'sils', but we recommend 'ma97' if it available.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

`unsymmetric_linear_solver` is a scalar variable of type default CHARACTER and length 30, that specifies the external package to be used to solve any unsymmetric linear system that might arise. Current possible choices are 'gls', 'ma28' and 'ma48'. See the documentation for the GALAHAD package ULS for further details. The default is `unsymmetric_linear_solver = 'gls'`, but we recommend 'ma48' if it available.

`prefix` is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string `prefix(2:LEN(TRIM(prefix))-1)`, thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default `prefix = ""`.

`SLS_control` is a scalar variable argument of type `SLS_control_type` that is used to pass control options to external packages used to factorize relevant symmetric matrices that arise. See the documentation for the GALAHAD package SLS for further details. In particular, default values are as for SLS, except that `SLS_control%relative_pivot_tolerance` is reset to `pivot_tol`.

`ULS_control` is a scalar variable argument of type `ULS_control_type` that is used to pass control options to external packages used to factorize relevant unsymmetric matrices that arise. See the documentation for the GALAHAD package ULS for further details. In particular, default values are as for ULS, except that `ULS_control%relative_pivot_tolerance` is reset to `pivot_tol`.

### 2.3.2 The derived data type for holding timing information

The derived data type `FDC_time_type` is used to hold elapsed CPU and system clock times for the various parts of the calculation. The components of `FDC_time_type` are:

`total` is a scalar variable of type `REAL(rp_)`, that gives the total CPU time spent in the package.

`analyse` is a scalar variable of type `REAL(rp_)`, that gives the CPU time spent analysing the required matrices prior to factorization.

`factorize` is a scalar variable of type `REAL(rp_)`, that gives the CPU time spent factorizing the required matrices.

`clock_total` is a scalar variable of type `REAL(rp_)`, that gives the total elapsed system clock time spent in the package.

`clock_analyse` is a scalar variable of type `REAL(rp_)`, that gives the elapsed system clock time spent analysing the required matrices prior to factorization.

`clock_factorize` is a scalar variable of type `REAL(rp_)`, that gives the elapsed system clock time spent factorizing the required matrices.

### 2.3.3 The derived data type for holding informational parameters

The derived data type `FDC_inform_type` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `FDC_inform_type` are:

`status` is a scalar variable of type `INTEGER(ip_)`, that gives the exit status of the algorithm. See Section 2.5 for details.

`alloc_status` is a scalar variable of type `INTEGER(ip_)`, that gives the status of the last attempted array allocation or deallocation. This will be 0 if `status = 0`.

`bad_alloc` is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if `status = 0`.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

`factorization_status` is a scalar variable of type `INTEGER(ip_)`, that gives the return status from the matrix factorization.

`factorization_integer` is a scalar variable of type `long INTEGER(ip_)`, that gives the amount of integer storage used for the matrix factorization.

`factorization_real` is a scalar variable of type `INTEGER(int64)`, that gives the amount of real storage used for the matrix factorization.

`non_negligible_pivot` is a scalar variable of type `REAL(rp_)`, that holds the value of the smallest pivot larger than `control%zero_pivot` when searching for dependent linear constraints. If `non_negligible_pivot` is close to `control%zero_pivot`, this may indicate that there are further dependent constraints, and it may be worth increasing `control%zero_pivot` above `non_negligible_pivot` and solving again.

`time` is a scalar variable of type `FDC_time_type` whose components are used to hold elapsed CPU and system clock times for the various parts of the calculation (see Section 2.3.2).

`SLS_inform` is a scalar variable argument of type `SLS_inform_type` that is used to pass information concerning the progress of the external packages used to factorize relevant symmetric matrices that arise. See the documentation for the GALAHAD package `SLS` for further details.

`ULS_inform` is a scalar variable argument of type `ULS_inform_type` that is used to pass information concerning the progress of the external packages used to factorize relevant symmetric matrices that arise. See the documentation for the GALAHAD package `ULS` for further details.

## 2.4 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.6 for further features):

1. The subroutine `FDC_initialize` is used to set default values before attempting to identify dependent constraints.
2. The subroutine `FDC_find_dependent` is called to identify dependent constraints.
3. The subroutine `FDC_terminate` is provided to allow the user to automatically deallocate workspace array components, previously allocated by `FDC_find_dependent`, after use.

We use square brackets [ ] to indicate OPTIONAL arguments.

### 2.4.1 The initialization subroutine

Default values are provided as follows:

```
CALL FDC_initialize( data, control, inform )
```

`data` is a scalar `INTENT(INOUT)` argument of type `FDC_data_type` that need not be set on entry and whose components will be used as workspace.

`control` is a scalar `INTENT(OUT)` argument of type `FDC_control_type` (see Section 2.3.1). On exit, `control` contains default values for the components as described in Section 2.3.1. These values should only be changed after calling `FDC_initialize`.

`inform` is a scalar `INTENT(INOUT)` argument of type `FDC_inform_type` (see Section 2.3.3). A successful call to `FDC_find_dependent` is indicated when the component status has the value 0. For other return values of status, see Section 2.5.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

### 2.4.2 The subroutine for finding dependent constraints

Dependent constraints are identified as follows:

```
CALL FDC_find_dependent( n, m, A_val, A_col, A_ptr, B, &
                        n_depen, DEPEND, data, control, inform )
```

**m** is a scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the number of constraints,  $m$ . **Restriction:**  $m \geq 0$ .

**n** is a scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the number of unknowns,  $n$ . **Restriction:**  $n \geq 0$ .

**A\_val** is an INTENT(IN) rank-one array of type REAL(rp\_), that holds the values of the entries (that is those component whose values are nonzero) of the matrix **A** whose rows are the vectors  $\mathbf{a}_i^T$ ,  $i = 1, \dots, m$ . The entries for row  $i$  must directly precede those in row  $i$ , but the order within each row is unimportant.

**A\_col** is an INTENT(IN) rank-one array of type INTEGER(ip\_), that holds the (column) indices of the entries of **A** corresponding to the values input in **A\_val**.

**A\_ptr** is an INTENT(IN) rank-one array of dimension  $m+1$  and type INTEGER(ip\_), whose  $i$ -th entry holds the starting position of row  $i$  of **A** for  $i = 1, \dots, m$ . The  $m+1$ -st entry of **A\_ptr** must hold the total number of entries plus one.

**B** is an INTENT(IN) rank-one array of dimension  $m$  and type REAL(rp\_), whose  $i$ -th component must be set to  $b_i$  for  $i = 1, \dots, m$ .

**n\_depen** is a scalar INTENT(OUT) argument of type INTEGER(ip\_), that gives the number of dependent constraints.

**DEPEND** is a rank-one allocatable array of type INTEGER(ip\_), On exit, if  $n\_depen > 0$ , it will have been allocated to be of length  $n\_depen$  and its components will be the indices of the dependent constraints. It will not be allocated or set if  $n\_depen = 0$ .

**data** is a scalar INTENT(INOUT) argument of type FDC\_data\_type that need not be set on entry and whose components will be used as workspace.

**control** is a scalar INTENT(IN) argument of type FDC\_control\_type (see Section 2.3.1). Default values may be assigned by calling FDC\_initialize prior to the first call to FDC\_find\_dependent.

**inform** is a scalar INTENT(INOUT) argument of type FDC\_inform\_type (see Section 2.3.3). A successful call to FDC\_find\_dependent is indicated when the component status has the value 0. For other return values of status, see Section 2.5.

### 2.4.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL FDC_terminate( data, control, inform[, C_depen ])
```

**data** is a scalar INTENT(INOUT) argument of type FDC\_data\_type whose array components will be deallocated on exit.

**control** is a scalar INTENT(IN) argument of type FDC\_control\_type exactly as for FDC\_find\_dependent.

**inform** is a scalar INTENT(INOUT) argument of type FDC\_inform\_type exactly as for FDC\_find\_dependent. Only the components status alloc\_status and bad\_alloc will be set on exit, and a successful call to FDC\_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.5.

**C\_depen** is an OPTIONAL rank-one allocatable array of type default INTEGER(ip\_), that will be deallocated on exit if PRESENT.

---

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

## 2.5 Warning and error messages

A negative value of `inform%status` on exit from `FDC_find_dependent` or `FDC_terminate` indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

- 1. An allocation error occurred. A message indicating the offending array is written on unit `control%error`, and the returned allocation status and a string containing the name of the offending array are held in `inform%alloc_status` and `inform%bad_alloc` respectively.
- 2. A deallocation error occurred. A message indicating the offending array is written on unit `control%error` and the returned allocation status and a string containing the name of the offending array are held in `inform%alloc_status` and `inform%bad_alloc` respectively. `status` is given by the value `inform%alloc_status`.
- 3. One of the restrictions  $n \geq 0$  or  $m \geq 0$  has been violated.
- 5 The constraints appear to be inconsistent.
- 9. The analysis phase of the factorization failed; the return status from the factorization package is given in the component `inform%factorization_status`.
- 10. The factorization failed; the return status from the factorization package is given in the component `inform%factorization_status`.

## 2.6 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable `control` of type `FDC_control_type` (see Section 2.3.1), by reading an appropriate data specification file using the subroutine `FDC_read_specfile`. This facility is useful as it allows a user to change FDC control parameters without editing and recompiling programs that call FDC.

A specification file, or `specfile`, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the `specfile` is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by `FDC_read_specfile` must start with a "BEGIN FDC" command and end with an "END" command. The syntax of the `specfile` is thus defined as follows:

```
( .. lines ignored by FDC_read_specfile .. )
BEGIN FDC
    keyword    value
    .....
    keyword    value
END
( .. lines ignored by FDC_read_specfile .. )
```

where `keyword` and `value` are two strings separated by (at least) one blank. The "BEGIN FDC" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN FDC SPECIFICATION
```

and

---

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

## END FDC SPECIFICATION

are acceptable. Furthermore, between the “BEGIN FDC” and “END” delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or \* are ignored. The content of a line after a ! or \* character is also ignored (as is the ! or \* character itself). This provides an easy manner to “comment out” some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are “ON”, “TRUE”, “.TRUE.”, “T”, “YES”, “Y”, or “OFF”, “NO”, “N”, “FALSE”, “.FALSE.” and “F”. Empty values are also allowed for logical control parameters, and are interpreted as “TRUE”.

The specification file must be open for input when `FDC_read_specfile` is called, and the associated device number passed to the routine in `device` (see below). Note that the corresponding file is `REWINDED`, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by `FDC_read_specfile`.

### 2.6.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL FDC_read_specfile( control, device )
```

`control` is a scalar `INTENT(INOUT)` argument of type `FDC_control_type` (see Section 2.3.1). Default values should have already been set, perhaps by calling `FDC_initialize`. On exit, individual components of `control` may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.3.1) of `control` that each affects are given in Table 2.1.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
maximum-permitted-infeasibility	%max_infeas	real
pivot-tolerance-used-for-dependencies	%pivot_tol	real
use-sls	%use_sls	logical
scale-A	%scale	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical
symmetric-linear-equation-solver	%symmetric_linear_solver	character
output-line-prefix	%prefix	character

Table 2.1: Specfile commands and associated components of `control`.

`device` is a scalar `INTENT(IN)` argument of type `INTEGER(ip_)`, that must be set to the unit number on which the specfile has been opened. If `device` is not open, `control` will not be altered and execution will continue, but an error message will be printed on unit `control%error`.

### 2.7 Information printed

If `control%print_level` is positive, information about the progress of the algorithm will be printed on unit `control%out`. If `control%print_level = 1`, basic statistics of the performance of the package will be produced. If `control%print_level ≥ 2` this output will be increased to provide details such as the size of each neglected pivot.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

### 3 GENERAL INFORMATION

**Use of common:** None.

**Workspace:** Provided automatically by the module.

**Other routines called directly:** None.

**Other modules used directly:** FDC\_find\_dependent calls the GALAHAD packages GALAHAD\_CLOCK, GALAHAD\_SYMBOLS, GALAHAD\_STRING, GALAHAD\_SMT, GALAHAD\_ROOTS, GALAHAD\_SLS, GALAHAD\_ULS, GALAHAD\_SPECFILE and GALAHAD\_SPACE.

**Input/output:** Output is under control of the arguments control%error, control%out and control%print\_level.

**Restrictions:**  $n \geq 0, m \geq 0$ .

**Portability:** ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

### 4 METHOD

A choice of two methods is available. In the first, the matrix

$$\mathbf{K} = \begin{pmatrix} \alpha \mathbf{I} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{pmatrix}$$

is formed and factorized for some small  $\alpha > 0$  using the GALAHAD package SLS—the factors  $\mathbf{K} = \mathbf{P}\mathbf{L}\mathbf{D}\mathbf{L}^T\mathbf{P}^T$  are used to determine whether  $\mathbf{A}$  has dependent rows. In particular, in exact arithmetic dependencies in  $\mathbf{A}$  will correspond to zero pivots in the block diagonal matrix  $\mathbf{D}$ .

The second choice of method finds factors  $\mathbf{A} = \mathbf{P}\mathbf{L}\mathbf{U}\mathbf{Q}$  of the rectangular matrix  $\mathbf{A}$  using the GALAHAD package ULS. In this case, dependencies in  $\mathbf{A}$  will be reflected in zero diagonal entries in  $\mathbf{U}$  in exact arithmetic.

The factorization in either case may also be used to determine whether the system is consistent.

### 5 EXAMPLE OF USE

Suppose we wish to find whether the linear constraints  $x_1 + 2x_2 + 3x_3 + 4x_4 = 5$ ,  $2x_1 - 4x_2 + 6x_3 - 8x_4 = 10$  and  $5x_2 + 10x_4 = 0$  are consistent but redundant. Then we may use the following code.

```
! THIS VERSION: GALAHAD 4.0 - 20/01/2022 AT 09:30 GMT.
PROGRAM GALAHAD_FDC_example
USE GALAHAD_FDC_double                                ! double precision version
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )           ! set precision
INTEGER, PARAMETER :: n = 4, m = 3, a_ne = 10
INTEGER :: A_ptr( m + 1 ), A_col( a_ne )
REAL ( KIND = wp ) :: A_val( a_ne ), B( m )
INTEGER, ALLOCATABLE :: DEPEND( : )
INTEGER :: n_depen
TYPE ( FDC_data_type ) :: data
TYPE ( FDC_control_type ) :: control
TYPE ( FDC_inform_type ) :: inform
A_val = (/ 1.0_wp, 2.0_wp, 3.0_wp, 4.0_wp, 2.0_wp, -4.0_wp, 6.0_wp,      &
          -8.0_wp, 5.0_wp, 10.0_wp /)
A_col = (/ 1, 2, 3, 4, 1, 2, 3, 4, 2, 4 /)
```

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

```
A_ptr = (/ 1, 5, 9, 11 /)
B = (/ 5.0_wp, 10.0_wp, 0.0_wp /)
CALL FDC_initialize( data, control, inform ) ! Initialize control parameters
control%use_sls = .TRUE.
control%symmetric_linear_solver = 'sytr'
CALL FDC_find_dependent( n, m, A_val, A_col, A_ptr, B, n_depen, DEPEND, &
                        data, control, inform ) ! Check for dependencies
IF ( inform%status == 0 ) THEN ! Successful return
  IF ( n_depen == 0 ) THEN
    WRITE( 6, "( ' FDC_find_dependent - no dependencies ' )" )
  ELSE
    WRITE( 6, "( ' FDC_find_dependent - dependent constraint(s):', 3I3 )" ) &
      DEPEND
  END IF
ELSE ! Error returns
  WRITE( 6, "( ' FDC_find_dependent exit status = ', I6 ) " ) inform%status
END IF
CALL FDC_terminate( data, control, inform, DEPEND ) ! Delete workspace
END PROGRAM GALAHAD_FDC_example
```

This produces the following output:

```
FDC_find_dependent - dependent constraint(s): 1
```